


```
LL      IIIIII  BBBB BBBB  FFFFFFFF  IIIIII  XX      XX  UU      UU  PPPPPPPP  FFFFFFFF
LL      IIIIII  BBBB BBBB  FFFFFFFF  IIIIII  XX      XX  UU      UU  PPPPPPPP  FFFFFFFF
LL      II      BB      BB  FF      FF  II      II  UU      UU  PP      PP  FF      FF
LL      II      BB      BB  FF      FF  II      II  UU      UU  PP      PP  FF      FF
LL      II      BB      BB  FF      FF  II      II  UU      UU  PP      PP  FF      FF
LL      II      BBBB BBBB  FFFFFFFF  II      II  XX      XX  UU      UU  PPPPPPPP  FFFFFFFF
LL      II      BBBB BBBB  FFFFFFFF  II      II  XX      XX  UU      UU  PPPPPPPP  FFFFFFFF
LL      II      BB      BB  FF      FF  II      II  XX      XX  UU      UU  PP      PP  FF      FF
LL      II      BB      BB  FF      FF  II      II  XX      XX  UU      UU  PP      PP  FF      FF
LL      II      BB      BB  FF      FF  II      II  XX      XX  UU      UU  PP      PP  FF      FF
LL      II      BB      BB  FF      FF  II      II  XX      XX  UU      UU  PP      PP  FF      FF
LLLLLLLL  IIIIII  BBBB BBBB  FF      FF  IIIIII  XX      XX  UUUUUUUUU  PP      PP  FF      FF
LLLLLLLL  IIIIII  BBBB BBBB  FF      FF  IIIIII  XX      XX  UUUUUUUUU  PP      PP  FF      FF
```

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLL  IIIIII  SSSSSSSS
```

(2) 60
(3) 257
(5) 485
(6) 729

DECLARATIONS

LIB\$FIXUP_FLT - Fixup floating reserved operand

NEXT_OPERAND - Get next operand and check for floating opcode

GET_REGS Get contents and addresses of all save registers in stack


```
0000 1 .TITLE LIB$FIXUP_FLT - Fixup floating reserved operand
0000 2 .IDENT 72-006/ ; File: LIBFIXUPF.MAR Edit: DGP2006
0000 3
0000 4
0000 5 *****
0000 6 *
0000 7 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 * ALL RIGHTS RESERVED.
0000 10 *
0000 11 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 * TRANSFERRED.
0000 17 *
0000 18 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 * CORPORATION.
0000 21 *
0000 22 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 *
0000 25 *
0000 26 *****
0000 27
0000 28
0000 29 ++
0000 30 FACILITY: General Utility Library
0000 31
0000 32 ABSTRACT:
0000 33
0000 34 LIB$FIXUP_FLT fixes up floating reserved operands when a
0000 35 reserved operand fault occurs so that the instruction may
0000 36 may be continued. It is designed to be a condition handler
0000 37 or to be called from a condition handler.
0000 38
0000 39 ENVIRONMENT: Runs at any access mode, AST Reentrant
0000 40
0000 41 AUTHOR: Thomas N. Hastings, Version 1, CREATION DATE: 06-AUG-1977
0000 42 Steven B. Lionel, Version 2, CREATION DATE: 02-JAN-1980
0000 43
0000 44 MODIFIED BY:
0000 45
0000 46 2-001 - Adapted from LIB$FIXUP_FLT version 1-004. Use table lookup for
0000 47 opcode and operand selection, remove PROBEs, improve operand
0000 48 fixup logic. SBL 2-JAN-1980
0000 49 2-002 - MOVAB of a constant isn't PIC, use MOVL instead. SBL 9-Jan-1980
0000 50 2-003 - Correct the MOVB at NEXT_OPERAND to a MOVZBL so that there isn't
0000 51 garbage in the upper part of R0. SBL 12-Jan-1981
0000 52 2-004 - Use local handler so that we only return SS$ ACCVIO and SS$ ROPRND
0000 53 as statuses, all other exceptions (like SS$ OPCDEC) resignal.
0000 54 SBL 13-Oct-1981
0000 55 2-005 - Use LIB$GET_OPCODE if BPT seen. SBL 4-Jan-1982
0000 56 2-006 - Defend the PROBE instruction by checking for SF$L_SAVE_REGS greater
0000 57 than 512. DGP 15-Mar-1982
```

LIB\$FIXUP_FLT
2-006

- Fixup floating reserved operand^{G 2}

16-SEP-1984 00:09:10 VAX/VMS Macro V04-00 Page 2
6-SEP-1984 11:07:14 [LIBRTL.SRC]LIB\$FIXUPF.MAR;1 (1)

0000 58 ;--

```
0000 60      .SBTTL  DECLARATIONS
0000 61      :
0000 62      : LIBRARY MACRO CALLS:
0000 63      :
0000 64      $SFDEF      : Stack frame symbols
0000 65      $CHFDEF     : Condition handling facility symbols
0000 66      $STSDEF     : Status value symbols
0000 67      $SSDEF      : System status values
0000 68      :
0000 69      : EXTERNAL DECLARATIONS:
0000 70      :
0000 71      .DSABL  GBL      : Force all external symbols to be declared
0000 72      .EXTRN  LIB$ BADSTA : Bad stack frame
0000 73      .EXTRN  LIB$SIG_TO_RET : Convert signals to return status
0000 74      .EXTRN  LIB$GET_OPCODE : Get debugger-modified opcode
0000 75      .EXTRN  SYSSCALL_HANDL : System routine that calls handlers
0000 76      :
0000 77      : MACROS:
0000 78      :
0000 79      NONE
0000 80      :
0000 81      : EQUATED SYMBOLS:
0000 82      :
00000000 0000 83      RO_OFF  = 0*4      : R0 register offset in register image
00000004 0000 84      R1_OFF  = 1*4      : R1 register offset
00000030 0000 85      AP_OFF  = 12*4     : AP register offset
00000034 0000 86      FP_OFF  = 13*4     : FP register offset
00000038 0000 87      SP_OFF  = 14*4     : SP register offset
0000003C 0000 88      PC_OFF  = 15*4     : PC register offset
00000040 0000 89      PSL_OFF = 16*4     : PSL offset
0000 90      :
00000044 0000 91      K_SAV_IMAGE_SIZ = 4*17 : size of each image vector
FFFFFFBC 0000 92      REG_IMAGE = -K_SAV_IMAGE_SIZ : FP offset for image vector of registers
FFFFFF78 0000 93      ADR_IMAGE = -<K_SAV_IMAGE_SIZ>*2 : FP offset for image vector of addresses
0000 94      : where registers have been saved in stack
FFFFFFFC 0000 95      IMAGE_PSL = -4      : FP offset of PSL image
FFFFFFF8 0000 96      IMAGE_PC = -8      : FP offset of PC image
0000 97      :
0000 98      :
0000 99      : Define first byte of two byte opcodes for G and H. G instructions
0000 100     : are fffD where ff is a F floating opcode and H instructions are
0000 101     : ddFD where dd is a D floating opcode. For example, POLYG is 55FD
0000 102     : and POLYH is 75FD.
0000 103     :
000000FD 0000 104     G_H      = ^XFD      : first G and H opcode byte
0000 105     :
00000003 0000 106     BPT      = ^X03      : Opcode for BPT instruction
0000 107     :
0000 108     : Define field in floating data types to test for reserved operand.
0000 109     :
00000009 0000 110     S_FMZERO = 9      : size for F
00000007 0000 111     V_FMZERO = 7      : position for F
0000000C 0000 112     S_GMZERO = 12     : size for G
00000004 0000 113     V_GMZERO = 4      : position for G
00000010 0000 114     S_HMZERO = 16     : size for H
00000000 0000 115     V_HMZERO = 0      : position for H
0000 116     :
```



```
0000 117 : Define codes used to denote operand types in opcode/operand tables
0000 118 : to follow.
0000 119
00000000 0000 120      OP_Z      = 0      : No more operands to process
00000001 0000 121      OP_B      = 1      : Byte
00000002 0000 122      OP_W      = 2      : Word
00000003 0000 123      OP_F      = 3      : F_floating
00000004 0000 124      OP_D      = 4      : D_floating
00000005 0000 125      OP_G      = 5      : G_floating
00000006 0000 126      OP_H      = 6      : H_floating
0000 127
0000 128 : OWN STORAGE:
0000 129
00000000 130      .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
0000 131      EXE, RD, NOWRT, LONG
0000 132
0000 133
0000 134 : Tables of opcodes and operand types. The first byte in each entry
0000 135 : is the opcode (or second byte of a 2-byte opcode). The remaining
0000 136 : bytes (up to 3) are OP_x codes defined above that specify what datatype
0000 137 : each operand is for that instruction. If an operand type is 0, then
0000 138 : no more operands are processed for that instruction.
0000 139
0000 140 : These tables are binary searched so the opcodes must be in ascending
0000 141 : order and the entry addresses must be longword aligned. This latter
0000 142 : requirement is met by having these tables be first in this module.
0000 143
0000 144 : Table for single byte opcodes
0000 145 $ING_TAB:
00 03 03 40 0000 146      .BYTE      ^X40, OP_F, OP_F, 0      : ADDF2
00 03 03 41 0004 147      .BYTE      ^X41, OP_F, OP_F, 0      : ADDF3
00 03 03 42 0008 148      .BYTE      ^X42, OP_F, OP_F, 0      : SUBF2
00 03 03 43 000C 149      .BYTE      ^X43, OP_F, OP_F, 0      : SUBF3
00 03 03 44 0010 150      .BYTE      ^X44, OP_F, OP_F, 0      : MULF2
00 03 03 45 0014 151      .BYTE      ^X45, OP_F, OP_F, 0      : MULF3
00 03 03 46 0018 152      .BYTE      ^X46, OP_F, OP_F, 0      : DIVF2
00 03 03 47 001C 153      .BYTE      ^X47, OP_F, OP_F, 0      : DIVF3
00 00 03 48 0020 154      .BYTE      ^X48, OP_F, 0, 0      : CVTFB
00 00 03 49 0024 155      .BYTE      ^X49, OP_F, 0, 0      : CVTFW
00 00 03 4A 0028 156      .BYTE      ^X4A, OP_F, 0, 0      : CVTFL
00 00 03 4B 002C 157      .BYTE      ^X4B, OP_F, 0, 0      : CVTRFL
03 03 03 4F 0030 158      .BYTE      ^X4F, OP_F, OP_F, OP_F      : ACBF
00 00 03 50 0034 159      .BYTE      ^X50, OP_F, 0, 0      : MOVF
00 03 03 51 0038 160      .BYTE      ^X51, OP_F, OP_F, 0      : CMPF
00 00 03 52 003C 161      .BYTE      ^X52, OP_F, 0, 0      : MNEGF
00 00 03 53 0040 162      .BYTE      ^X53, OP_F, 0, 0      : TSTF
03 01 03 54 0044 163      .BYTE      ^X54, OP_F, OP_B, OP_F      : EMOF
00 00 03 55 0048 164      .BYTE      ^X55, OP_F, 0, 0      : POLYF
00 00 03 56 004C 165      .BYTE      ^X56, OP_F, 0, 0      : CVTFD
00 04 04 60 0050 166      .BYTE      ^X60, OP_D, OP_D, 0      : ADDD2
00 04 04 61 0054 167      .BYTE      ^X61, OP_D, OP_D, 0      : ADDD3
00 04 04 62 0058 168      .BYTE      ^X62, OP_D, OP_D, 0      : SUBD2
00 04 04 63 005C 169      .BYTE      ^X63, OP_D, OP_D, 0      : SUBD3
00 04 04 64 0060 170      .BYTE      ^X64, OP_D, OP_D, 0      : MULD2
00 04 04 65 0064 171      .BYTE      ^X65, OP_D, OP_D, 0      : MULD3
00 04 04 66 0068 172      .BYTE      ^X66, OP_D, OP_D, 0      : DIVD2
00 04 04 67 006C 173      .BYTE      ^X67, OP_D, OP_D, 0      : DIVD3
```

```
00 00 04 68 0070 174 .BYTE ^X68, OP_D, 0, 0 : CVTDB
00 00 04 69 0074 175 .BYTE ^X69, OP_D, 0, 0 : CVTDW
00 00 04 6A 0078 176 .BYTE ^X6A, OP_D, 0, 0 : CVTDL
00 00 04 6B 007C 177 .BYTE ^X6B, OP_D, 0, 0 : CVTRDL
04 04 04 6F 0080 178 .BYTE ^X6F, OP_D, OP_D, OP_D : ACBD
00 00 04 70 0084 179 .BYTE ^X70, OP_D, 0, 0 : MOVD
00 04 04 71 0088 180 .BYTE ^X71, OP_D, OP_D, 0 : CMPD
00 00 04 72 008C 181 .BYTE ^X72, OP_D, 0, 0 : MNEGD
00 00 04 73 0090 182 .BYTE ^X73, OP_D, 0, 0 : TSTD
04 01 04 74 0094 183 .BYTE ^X74, OP_D, OP_B, OP_D : EMODD
00 00 04 75 0098 184 .BYTE ^X75, OP_D, 0, 0 : POLYD
00 00 04 76 009C 185 .BYTE ^X76, OP_D, 0, 0 : CVTDF
00 00 04 76 00A0 186 SING_END:
00 00 04 76 00A0 187
00 00 04 76 00A0 188 ; Table for 2-byte opcodes. First byte (^XFD) is omitted.
00 00 04 76 00A0 189
00 00 04 76 00A0 190 DOUB_TAB:
00 00 04 32 00A0 191 .BYTE ^X32, OP_D, 0, 0 : CVTDH
00 00 05 33 00A4 192 .BYTE ^X33, OP_G, 0, 0 : CVTGF
00 05 05 40 00A8 193 .BYTE ^X40, OP_G, OP_G, 0 : ADDG2
00 05 05 41 00AC 194 .BYTE ^X41, OP_G, OP_G, 0 : ADDG3
00 05 05 42 00B0 195 .BYTE ^X42, OP_G, OP_G, 0 : SUBG2
00 05 05 43 00B4 196 .BYTE ^X43, OP_G, OP_G, 0 : SUBG3
00 05 05 44 00B8 197 .BYTE ^X44, OP_G, OP_G, 0 : MULG2
00 05 05 45 00BC 198 .BYTE ^X45, OP_G, OP_G, 0 : MULG3
00 05 05 46 00C0 199 .BYTE ^X46, OP_G, OP_G, 0 : DIVG2
00 05 05 47 00C4 200 .BYTE ^X47, OP_G, OP_G, 0 : DIVG3
00 00 05 48 00C8 201 .BYTE ^X48, OP_G, 0, 0 : CVTGB
00 00 05 49 00CC 202 .BYTE ^X49, OP_G, 0, 0 : CVTGW
00 00 05 4A 00D0 203 .BYTE ^X4A, OP_G, 0, 0 : CVTGL
00 00 05 4B 00D4 204 .BYTE ^X4B, OP_G, 0, 0 : CVTRGL
05 05 05 4F 00D8 205 .BYTE ^X4F, OP_G, OP_G, OP_G : ACBG
00 00 05 50 00DC 206 .BYTE ^X50, OP_G, 0, 0 : MOVG
00 05 05 51 00E0 207 .BYTE ^X51, OP_G, OP_G, 0 : CMPG
00 00 05 52 00E4 208 .BYTE ^X52, OP_G, 0, 0 : MNEGG
00 00 05 53 00E8 209 .BYTE ^X53, OP_G, 0, 0 : TSTG
05 02 05 54 00EC 210 .BYTE ^X54, OP_G, OP_W, OP_G : EMODG
00 00 05 55 00F0 211 .BYTE ^X55, OP_G, 0, 0 : POLYG
00 00 05 56 00F4 212 .BYTE ^X56, OP_G, 0, 0 : CVTGH
00 06 06 60 00F8 213 .BYTE ^X60, OP_H, OP_H, 0 : ADDH2
00 06 06 61 00FC 214 .BYTE ^X61, OP_H, OP_H, 0 : ADDH3
00 06 06 62 0100 215 .BYTE ^X62, OP_H, OP_H, 0 : SUBH2
00 06 06 63 0104 216 .BYTE ^X63, OP_H, OP_H, 0 : SUBH3
00 06 06 64 0108 217 .BYTE ^X64, OP_H, OP_H, 0 : MULH2
00 06 06 65 010C 218 .BYTE ^X65, OP_H, OP_H, 0 : MULH3
00 06 06 66 0110 219 .BYTE ^X66, OP_H, OP_H, 0 : DIVH2
00 06 06 67 0114 220 .BYTE ^X67, OP_H, OP_H, 0 : DIVH3
00 00 06 68 0118 221 .BYTE ^X68, OP_H, 0, 0 : CVTHB
00 00 06 69 011C 222 .BYTE ^X69, OP_H, 0, 0 : CVTHW
00 00 06 6A 0120 223 .BYTE ^X6A, OP_H, 0, 0 : CVTHL
00 00 06 6B 0124 224 .BYTE ^X6B, OP_H, 0, 0 : CVTRHL
06 06 06 6F 0128 225 .BYTE ^X6F, OP_H, OP_H, OP_H : ACBH
00 00 06 70 012C 226 .BYTE ^X70, OP_H, 0, 0 : MOVH
00 06 06 71 0130 227 .BYTE ^X71, OP_H, OP_H, 0 : CMPH
00 00 06 72 0134 228 .BYTE ^X72, OP_H, 0, 0 : MNEGH
00 00 06 73 0138 229 .BYTE ^X73, OP_H, 0, 0 : TSTH
06 02 06 74 013C 230 .BYTE ^X74, OP_H, OP_W, OP_H : EMODH
```



```
00 00 06 75 0140 231      .BYTE  ^X75, OP_H, 0 , 0      : POLYH
00 00 06 76 0144 232      .BYTE  ^X76, OP_H, 0 , 0      : CVTHG
00 00 03 98 0148 233      .BYTE  ^X98, OP_F, 0 , 0      : CVTFH
00 00 03 99 014C 234      .BYTE  ^X99, OP_F, 0 , 0      : CVTFG
00 00 06 F6 0150 235      .BYTE  ^XF6, OP_H, 0 , 0      : CVTHF
00 00 06 F7 0154 236      .BYTE  ^XF7, OP_H, 0 , 0      : CVTHD
      0158 237 DOUB_END:
      0158 238
      0158 239 ; Table of operand sizes listed in OP_x code order
      0158 240
      0158 241 OP_SIZES:
00 0158 242      .BYTE  0      : Not used
01 0159 243      .BYTE  1      : Byte
02 015A 244      .BYTE  2      : Word
04 015B 245      .BYTE  4      : F_floating
08 015C 246      .BYTE  8      : D_floating
08 015D 247      .BYTE  8      : G_floating
10 015E 248      .BYTE 16      : H_floating
      015F 249
      015F 250 :
      015F 251 : PSECT DECLARATIONS:
      015F 252 :
      015F 253      .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
0000 015F 254      EXE, RD, NOWRT, LONG
      015F 255
```

```
015F 257 .SBTTL LIB$FIXUP_FLT - Fixup floating reserved operand
015F 258 :++
015F 259 : FUNCTIONAL DESCRIPTION:
015F 260 :
015F 261 : LIB$FIXUP_FLT finds the reserved operand of any F, D, G or H
015F 262 : floating instruction (with exceptions stated below) after a
015F 263 : reserved operand fault has been signaled. LIB$FIXUP_FLT changes
015F 264 : the reserved operand from -0.0 to the parameter, new_operand, is
015F 265 : present; or to +0.0 if new_operand is absent.
015F 266 :
015F 267 :
015F 268 :
015F 269 : Exceptions:
015F 270 :
015F 271 : LIB$FIXUP_FLT can not handle the following cases and will return
015F 272 : a status of SSS_RESIGNAL if any of them occur.
015F 273 :
015F 274 : 1. The currently active signaled condition is not
015F 275 : SSS_ROPRAND.
015F 276 :
015F 277 : 2. The reserved operand's datatype is not F, D, G or H
015F 278 : floating.
015F 279 :
015F 280 : 3. The reserved operand is an element in a POLYx
015F 281 : table.
015F 282 :
015F 283 : CALLING SEQUENCE:
015F 284 :
015F 285 : ret_status.wlc.v = LIB$FIXUP_FLT (sig_args_adr.rl.ra,
015F 286 : mch_args_adr.rl.ra [, new_operand.rf.r])
015F 287 :
015F 288 : FORMAL PARAMETERS:
015F 289 :
00000004 015F 290 : sig_args_adr = 4
015F 291 : Address of signal argument vector.
00000008 015F 292 : mch_args_adr = 8
015F 293 : Address of mechanism argument vector.
0000000C 015F 294 : new_operand = 12
015F 295 : Optional. Address of an F_floating value to replace the
015F 296 : reserved operand.
015F 297 :
015F 298 : IMPLICIT INPUTS:
015F 299 :
015F 300 : The stack frames back to that of the instruction which faulted.
015F 301 : The instruction which faulted and its operands.
015F 302 :
015F 303 : IMPLICIT OUTPUTS:
015F 304 :
015F 305 : The reserved floating operand, if found, is replaced by
015F 306 : 'new_operand' or zero.
015F 307 :
015F 308 : COMPLETION STATUS:
015F 309 :
015F 310 : SSS_CONTINUE - continue execution at point of condition
015F 311 : Routine successfully completed. The reserved operand was
015F 312 : found and was fixed up.
015F 313 : SSS_ACCVIO - access violation
```

```
015F 314 : An argument to LIB$FIXUP_FLT or an operand of the faulting
015F 315 : instruction could not be read or written.
015F 316 : $$$_RESIGNAL - resignal condition to next handler
015F 317 : The condition signaled was not $$$_ROPRAND or the reserved
015F 318 : operand was not a floating point value or was an element in
015F 319 : a POLYx table.
015F 320 : $$$_ROPRAND - reserved operand fault
015F 321 : The optional argument new_operand was supplied but was
015F 322 : itself an F floating reserved operand.
015F 323 : LIB$_BADSTA - bad stack
015F 324 : The stack frame linkage had been corrupted since the time of
015F 325 : the reserved operand exception.
015F 326 :
015F 327 : Note: If the status value returned from LIB$FIXUP_FLT is seen by
015F 328 : the condition handling facility, (as would be the case if
015F 329 : LIB$FIXUP_FLT was the handler), any success value is equivalent
015F 330 : to $$$_CONTINUE, which causes the instruction to be restarted.
015F 331 : Any failure value is equivalent to $$$_RESIGNAL, which will cause
015F 332 : the condition to be resignalled to the next handler. This is
015F 333 : because the condition handler (LIB$FIXUP_FLT) failed to handle
015F 334 : the condition correctly.
015F 335 :
015F 336 : SIDE EFFECTS:
015F 337 :
015F 338 : If the reserved operand is fixed up, the instruction which
015F 339 : faulted is restarted.
015F 340 :
015F 341 : --
015F 342 :
015F 343 : +
015F 344 : Registers used:
015F 345 :
015F 346 : R0 = scratch
015F 347 : R1 = scratch
015F 348 : R2 = pointer into opcode/operand table
015F 349 : R3 = context index or 0
015F 350 : R4 = OA1 (operand address) of bits 31:0
015F 351 : R5 = OA2 (operand address) of bits 63:32 which may not be
015F 352 : OA1+4 since registers not necessarily saved contiguously.
015F 353 : R6 = register number of operand specifier
015F 354 : R7 = scratch
015F 355 : R8 = OA3 (operand address) of bits 95:64 (H opcodes only)
015F 356 : R9 = OA4 (operand address) of bits 127:96 (H opcodes only)
015F 357 :
015F 358 :
015F 359 : .ENTRY LIB$FIXUP_FLT, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
0161 360 : save all registers so that all will be
0161 361 : found in stack during back scan.
0161 362 : disable IV (content index multiply)
0161 363 : ASSUME SF$$_SAVE_REGS LE 512 : Assembly time check to be sure that the
0161 364 : PROBES later will work correctly
0161 365 : MOVAB W^OUR_HANDLER, (FP) : Enable condition handler
0166 366 : MOVL SIG_ARGS_ADR(AP), R0 : R0 = adr. of signal arg list array
016A 367 : CMPZV #ST$$_COND_ID, - : position of message identification
0174 368 : #ST$$_COND_ID, - : size of id
0174 369 : CH$$_SIG_NAME(R0), - : compare 29-bit VAX-11 signal code
0174 370 : #<$$$_ROPRAND-$$$_COND_ID> : with reserved operand code
```

OFFC

```
0000008A 8F 04 A0 19 03 9E DO ED
6D 50 03EE'CF 04 AC
```

LIB\$
Sym
ADR
AP
AUT
AUT
AUT
BAD
BPT
BYT
BYT
CHF
CHF
CHF
CHF
CHF
CHF
CHK
DES
DES
DES
DES
DIS
DIS
DOU
DOU
END
EXT
FIX
GET
G H
IM
IND
K S
LIB
LIB
LIB
LIB
LIB
LONG
LONG
LOO
LOO
LOO
LOO
MCH
NEW
NEX
NEX
NEX
NEX
NOT
OP
OP
OP
OP
OP
OP
OUR


```
03 13 0174 371 BEQL 2$ ; It is, continue
0073 31 0176 372 BRW RESIGNAL ; Not, so resignal
SE FF78 CE 9E 0179 373 2$: MOVAB -<K_SAV_IMAGE_SIZ>*2(SP), SP ; allocate two local vectors:
; REG_IMAGE - image of registers at time of
; ADR_IMAGE - image of address where regs ar
; saved in stack in case they need fixup.
; setup the two image vectors in local stora
; do not return here if error, instead RET w
; error completion status
01DA 30 017E 376 BSBW GET_REGS
0181 378
0181 379
0181 380
0181 381
0181 382
0181 383
0181 384
0181 385
0181 386
0181 387 BSBW NEXT_BYTE ; Get first opcode byte
FD 8F 50 91 0184 388 CMPB R0, #G_H ; Is this a G or H instruction?
27 13 0188 389 BEQL 10$ ; Yes
03 50 91 018A 390 CMPB R0, #BPT ; Is this a BPT instruction?
12 12 018D 391 BNEQ 5$ ; Skip if not
7E FB AD 01 C3 018F 392 SUBL3 #1, IMAGE_PC(FP), -(SP) ; Push PC on stack
00000000 GF 01 FB 0194 393 CALLS #1, G^LIB$GET_OPCODE ; See what opcode really is.
FD 8F 50 91 019B 394 CMPB R0, #G_H ; Is it a G or H instruction?
10 13 019F 395 BEQL 10$ ; Skip if so
57 000000A0 8F D4 01A1 396 5$: CLRL R1 ; Binary search low limit
53 FES2 CF 9E 01AA 397 MOVL #<SING_END-SING_TAB>, R7 ; Binary search high limit
11 11 01AF 398 MOVAB W^SING_TAB, R3 ; Table base
018D 30 01B1 400 10$: BSBW NEXT_BYTE ; Get second opcode byte
51 D4 01B4 401 CLRL R1 ; Binary search low limit
57 000000B8 8F D0 01B6 402 MOVL #<DOUB_END-DOUB_TAB>, R7 ; Binary search high limit
53 FEDF CF 9E 01BD 403 MOVAB W^DOUB_TAB, R3 ; Table base
01C2 404
57 51 D1 01C2 405 SEARCH: CMPL R1, R7 ; Not in table?
25 13 01C5 406 BEQL RESIGNAL ; Yes, resignal
52 57 51 C1 01C7 407 ADDL3 R1, R7, R2 ; Get middle entry
52 52 FF 8F 78 01CB 408 ASHL #-1, R2, R2
52 52 03 CA 01D0 409 BICL2 #3, R2
50 6342 91 01D3 410 CMPB (R3)[R2], R0 ; Longword offset
08 1A 01D7 411 BGTRU 10$ ; Is this the opcode?
0B 1F 01D9 412 BLSSU 20$ ; No, too high
52 53 C0 01DB 413 ADDL2 R3, R2 ; No, too low
0011 31 01DE 414 BRW SCAN ; Compute actual address
57 52 D0 01E1 415 10$: MOVL R2, R7 ; Now scan the operands
DC 11 01E4 416 BRB SEARCH ; New high limit is last try
51 52 04 C1 01E6 417 20$: ADDL3 #4, R2, R1 ; Continue search
D6 11 01EA 418 BRB SEARCH ; New low limit is last try + 4
01EC 419
01EC 420
01EC 421
01EC 422
01EC 423
01EC 424
01EC 425
01EC 426
01F1 427
; If we can't handle this exception, return $$$_RESIGNAL.
RESIGNAL:
MOVZWL $$$_RESIGNAL, R0 ; R0 = RESIGNAL error completion code
RET ; error return
```

```
01F2 428
01F2 429 :+
01F2 430 : Now scan the operand list, looking for a reserved operand.
01F2 431 :-
01F2 432
01F2 433 SCAN:
52 52 D6 01F2 434 INCL R2 : Get next operand type byte
03 03 93 01F4 435 BITB #3, R2 : If the low two bits are clear,
: then we are at the start of the
: next table entry, thus we are
: done. Remember that the tables
: are longword aligned.
F3 13 01F7 436 : No reserved operand found
62 95 01F7 437 : No more operands to test?
EF 13 01F7 438 : Yes, no reserved operand found
0052 30 01F7 439 : Look at next operand
EF 50 E9 01F9 440 BEQL RESIGNAL : If reserved operand not found,
: continue looking
00 11 0200 441 TSTB (R2) : Fixup reserved operand
0203 442 BEQL RESIGNAL
0203 443 BSBW NEXT_OPERAND
BRB 444 BLBC R0, SCAN
FIXUP 445
446
```

```
0205 448 :+
0205 449 : Fixup reserved operand
0205 450 :-
0205 451
0205 452 FIXUP:
0205 453 CLRL R0 ; default fixup value is zero
0207 454 (AP), #<NEW_OPERAND/4> ; is new operand present?
020A 455 10$ ; no
020C 456 MOVL NEW_OPERAND(AP), R0 ; yes, get address
0210 457 BEQL 10$ ; omitted by reference?
0212 458 MOVL (R0), R0 ; get F_floating fixup value
0215 459 10$: CASEB (R2), #OP_F, #<OP_H-OP_F> ; select on operand type
0219 460 11$: .WORD DEST_F-11$ ; F_floating
021B 461 .WORD DEST_D-11$ ; D_floating
021D 462 .WORD DEST_G-11$ ; G_floating
021F 463 .WORD DEST_H-11$ ; H_floating
0221 464 HALT ; should never get here
0222 465 DEST_F: MOVF R0, (R4) ; move F_floating
0225 466 BRB EXIT
0227 467 DEST_D: CVTFD R0, -(SP) ; convert F to D and store
022A 468 MOVL (SP)+, (R4)
022D 469 MOVL (SP)+, (R5)
0230 470 BRB EXIT
0232 471 DEST_G: CVTFG R0, -(SP) ; convert F to G and store
0236 472 MOVL (SP)+, (R4)
0239 473 MOVL (SP)+, (R5)
023C 474 BRB EXIT
023E 475 DEST_H: CVTFH R0, -(SP) ; convert F to H and store
0242 476 MOVL (SP)+, (R4)
0245 477 MOVL (SP)+, (R5)
0248 478 MOVL (SP)+, (R8)
024B 479 MOVL (SP)+, (R9)
024E 480
024E 481 EXIT:
024E 482 MOVL #SS$_CONTINUE, R0 ; success
0251 483 RET ; return
```



```
0252 485 .SBTTL NEXT_OPERAND - Get next operand and check for floating opcode
0252 486 :++
0252 487 : FUNCTIONAL DESCRIPTION:
0252 488 :
0252 489 : NEXT_OPERAND interprets the instruction stream and
0252 490 : gets the next operand. It returns 1 in R0
0252 491 : if operand is floating or double reserved operand, else 0.
0252 492 :
0252 493 : CALLING SEQUENCE
0252 494 :
0252 495 : JSB NEXT_OPERAND
0252 496 :
0252 497 : INPUT PARAMETERS:
0252 498 :
0252 499 : R2 = address of operand type table
0252 500 :
0252 501 : IMPLICIT INPUTS:
0252 502 :
0252 503 : REG_IMAGE(FP) ; The image of the registers including PC
0252 504 : instruction stream
0252 505 :
0252 506 : OUTPUT PARAMETERS:
0252 507 :
0252 508 : R4 = OA1 (operand address of bits 31:0 of operand)
0252 509 : R5 = OA2 (operand address of bits 63:32 of operand) if R1 = 8
0252 510 : R8 = OA3 (H opcodes only)
0252 511 : R9 = OA4 (H opcodes only)
0252 512 :
0252 513 : IMPLICIT OUTPUT:
0252 514 :
0252 515 : Saved image of PC is updated as operand specific is interpreted
0252 516 :
0252 517 : COMPLETION STATUS
0252 518 :
0252 519 : R0 = 1 if operand is floating or double reserved operand, else 0
0252 520 :
0252 521 : SIDE EFFECTS:
0252 522 :
0252 523 : NONE - uses registers R0:R9 - see LIB$FIXUP_FLT for register usage
0252 524 :--
0252 525 :
0252 526 : NEXT_OPERAND:
0252 527 : CLRL R3 ; R3 = initial context index register
0252 528 : MOVZBL (R2), R0 ; Get operand type byte
0252 529 : MOVZBL W^OP_SIZES[R0], R1 ; Get operand size
0252 530 :
0252 531 : :+
0252 532 : : Loop to get operand specifier - loop back here (once) if operand specifier is inde
0252 533 : :--
0252 534 :
0252 535 : LOOP_OP:
0252 536 : BSBW NEXT_BYTE ; R0 = next I-stream byte (sign extended)
0252 537 : EXTZV #0, #4, R0, R6 ; R6 = register field
0252 538 : EXTZV #4, #4, R0, R0 ; R0 = operand specifier 7:4
0252 539 : CASEB R0, #4, #15-4 ; dispatch on operand specifier code
0252 540 : ; literal 0-3 falls through
0252 541 : 10$: .WORD INDEXED-10$ ; 4
```

51 50 53 D4
FEFC CF40 9A 9A

56 50 04 00 EF
50 50 04 04 EF
0B 04 50 8F

00E1 30
001B' 026E

```
0023: 0270 542 .WORD REG-10$ : 5
0038: 0272 543 .WORD REG_DEF-10$ : 6
0042: 0274 544 .WORD AUTO_DECR-10$ : 7
0049: 0276 545 .WORD AUTO_INCR-10$ : 8
0055: 0278 546 .WORD AUTO_INCR_DEF-10$ : 9
0064: 027A 547 .WORD BYTE_DISP[-10$ : 10
0068: 027C 548 .WORD BYTE_DISPL_DEF-10$ : 11
006C: 027E 549 .WORD WORD_DISPL-10$ : 12
0070: 0280 550 .WORD WORD_DISPL_DEF-10$ : 13
0074: 0282 551 .WORD LONG_DISPL-10$ : 14
007E: 0284 552 .WORD LONG_DISPL_DEF-10$ : 15
      0286 553
      0286 554
      0286 555 :+ Literal - can't be reserved, just return failure
      0286 556 :-
      00B5 31 0286 557
      0289 558 BRW NOT_RESRV ; return - not reserved operand
      0289 559
      0289 560 :+ Indexed - save context index and loop back
      0289 561 :-
      0289 562
      0289 563 INDEXED:
      53 51 BC AD46 C5 0289 564 MULL3 REG_IMAGE(FP)[R6], R1,- ; R3 = context index
      CC 11 028F 566 R3
      028F 567 BRB LOOP_OP ; go back and get next specifier
      0291 568
      0291 569 :+ Register
      0291 570 :-
      0291 571
      0291 572
      54 FF78 CD46 D0 0291 573 REG: MOVL ADR_IMAGE(FP)[R6], R4 ; R4 = OA1 = adr where Rn saved in stack
      55 FF7C CD46 D0 0297 574 MOVL ADR_IMAGE+4(FP)[R6], R5 ; R5 = OA2 = adr where Rn+1 saved in stack
      58 80 AD46 D0 029D 575 MOVL ADR_IMAGE+8(FP)[R6], R8 ; R8 = OA3
      59 84 AD46 D0 02A2 576 MOVL ADR_IMAGE+12(FP)[R6], R9 ; R9 = OA4
      SD 11 02A7 577 BRB CHK_OP_RSB ; check operand for reserved and RSB
      02A9 578
      02A9 579 :+ Register Deferred
      02A9 580 :-
      02A9 581
      02A9 582
      54 BC AD46 D0 02A9 583 REG_DEF: MOVL REG_IMAGE(FP)[R6], R4 ; R4 = OA = contents of Rn
      47 11 02AE 585 BRB SET_OA2 ; set OA2, check op and RSB
      02B0 586
      02B0 587 :+ Auto Decrement
      02B0 588 :-
      02B0 589
      02B0 590
      BC AD46 51 C2 02B0 591 AUTO_DECR:
      F2 11 02B5 592 SUBL R1, REG_IMAGE(FP)[R6] ; decrement Rn by operand size
      02B7 593 BRB REG_DEF ; go do register deferred
      02B7 594
      02B7 595 :+ Auto Increment
      02B7 596 :-
      02B7 597
      02B7 598
```

```
54 BC AD46 D0 02B7 599 AUTO_INCR:
BC AD46 51 C0 02B7 600      MOVL  REG_IMAGE(FP)[R6], R4      ; R4 = OA = contents of Rn
34 11 02BC 601      ADDL  R1, REG_IMAGE(FP)[R6]      ; increment Rn by operand size
02C1 602      BRB  SET_OA2      ; set OA2, check op and RSB
02C3 603
02C3 604      ;+
02C3 605      ; Auto Increment Deferred
02C3 606      ; -
02C3 607
02C3 608 AUTO_INCR DEF:
02C3 609      MOVL  REG_IMAGE(FP)[R6], R4      ; R4 = contents of Rn
54 BC AD46 D0 02C8 610      MOVL  (R4), R4      ; R4 = OA
54 54 64 D0 02CB 611      ADDL  #4, REG_IMAGE(FP)[R6]      ; increment Rn by 4 (size of address)
BC AD46 04 C0 02D0 612      BRB  SET_OA2      ; set OA2, check op, and RSB
25 11 02D2 613
02D2 614      ;+
02D2 615      ; Byte Displacement
02D2 616      ; -
02D2 617
02D2 618 BYTE_DISPL:
6D 10 02D2 619      BSBB  NEXT_BYTE      ; R0 = next I-stream byte
OE 11 02D4 620      BRB  DISPL      ; add to PC
02D6 621
02D6 622      ;+
02D6 623      ; Byte Displacement Deferred
02D6 624      ; -
02D6 625
02D6 626 BYTE_DISPL DEF:
69 10 02D6 627      BSBB  NEXT_BYTE      ; R0 = next I-stream byte
14 11 02D8 628      BRB  DISPL_DEF      ; add to PC and defer
02DA 629
02DA 630      ;+
02DA 631      ; Word Displacement
02DA 632      ; -
02DA 633
02DA 634 WORD_DISPL:
6D 10 02DA 635      BSBB  NEXT_WORD      ; R0 = next I-stream word
06 11 02DC 636      BRB  DISPL      ; add to PC
02DE 637
02DE 638      ;+
02DE 639      ; Word Displacement Deferred
02DE 640      ; -
02DE 641
02DE 642 WORD_DISPL DEF:
69 10 02DE 643      BSBB  NEXT_WORD      ; R0 = next I-stream word
0C 11 02E0 644      BRB  DISPL_DEF      ; add to PC and defer
02E2 645
02E2 646      ;+
02E2 647      ; Long displacement
02E2 648      ; -
02E2 649
02E2 650 LONG_DISPL:
54 BC AD46 6E 10 02E2 651      BSBB  NEXT_LONG      ; R0 = next I-stream longword
50 C1 02E4 652 DISPL: ADDL3 R0, REG_IMAGE(FP)[R6], R4      ; R4 = OA = (Rn) + displacement
0B 11 02EA 653      BRB  SET_OA2      ; set OA2, check OP, and RSB
02EC 654
02EC 655      ;+
```



```

54 BC AD46 50 C1 02EC 656 : Long Displacement deferred
54 54 64 D0 02EC 657 :-
02EC 658
02EC 659 LONG_DISPL DEF:
02EC 660 BSBB NEXT_LONG : R0 = Next I-stream longword
02EE 661 DISPL_DEF: : here for displacement deferred
02EE 662 ADDL3 R0, REG_IMAGE(FP)[R6], R4 : R4 = (Rn) + displacement
02F4 663 MOVL (R4), R4 : R4 = OA = (OA) (do defer)
02F7 664
02F7 665
02F7 666 :+
02F7 667 : add context index or 0
02F7 668 : Set OA2 (operand address 2) from OA+4 since
02F7 669 : operand is in memory not a register and therefore is contiguous
02F7 670 : Also set OA3 and OA4
02F7 671 :-
02F7 672 SET_OA2:
02F7 673 ADDL R3, R4 : R4 = OA + context index or 0
02FA 674 ADDL3 R4, #4, R5 : R5 = OA2 = OA + 4
02FE 675 ADDL3 R5, #4, R8 : R8 = OA3
0302 676 ADDL3 R8, #4, R9 : R9 = OA4
0306 677
0306 678 :+
0306 679 : check for reserved operand
0306 680 :-
0306 681
0306 682 CHK_OP_RSB:
0306 683 MOVL #1, R0 : Indicate success, initially
0309 684 CASEB (R2), #OP_F, #<OP_H-OP_F> : Case on operand type
030D 685 1$: .WORD 10$-1$ : F_floating
030F 686 .WORD 10$-1$ : D_floating
0311 687 .WORD 20$-1$ : G_floating
0313 688 .WORD 30$-1$ : H_floating
0315 689 BRB NOT RESRV : wrong datatype
0317 690 10$: CMPZV #V_FMZERO, #S_FMZERO, - : Check F_floating and D_floating
0318 691 (R4), #*X100
0320 692 BEQL NEXT_OPERANDX : Found
0322 693 BRB NOT RESRV : Not found
0324 694 20$: CMPZV #V_GMZERO, #S_GMZERO, - : Check G_floating
0328 695 (R4), #*X800
032D 696 BEQL NEXT_OPERANDX : Found
032F 697 BRB NOT RESRV : Not found
0331 698 30$: CMPZV #V_HMZERO, #S_HMZERO, - : Check H_floating
0335 699 (R4), #*X8000
033A 700 BEQL NEXT_OPERANDX : Found
033C 701 BRB NOT_RESRV : Not found
033E 702 NOT_RESRV:
033E 703 CLRL R0 : R0 = failure
0340 704 NEXT_OPERANDX:
0340 705 RSB : return R0 indicating success or failure
0341 706
0341 707
0341 708
0341 709 :+
0341 710 : routines to get next byte, word, or long from I-stream and sign extend
0341 711 :-
0341 712
```

```
50  F8 BD  98 0341 713 NEXT_BYTE:
    F8 AD  D6 0341 714 CVTBL @IMAGE_PC(FP), R0 ; R0 = next byte
    05 0345 715 INCL IMAGE_PC(FP) ; update PC
    0348 716 RSB ; return
    0349 717
50  F8 BD  32 0349 718 NEXT_WORD:
    F8 AD  02 C0 0349 719 CVTWL @IMAGE_PC(FP), R0 ; R0 = next word
    05 034D 720 ADDL #2, IMAGE_PC(FP) ; update PC
    0351 721 RSB ; return
    0352 722
50  F8 BD  D0 0352 723 NEXT_LONG:
    F8 AD  04 C0 0352 724 MOVL @IMAGE_PC(FP), R0 ; R0 = next longword
    05 0356 725 ADDL #4, IMAGE_PC(FP) ; update PC
    035A 726 RSB ; return
    035B 727
```

```
0358 729 .SBTTL GET_REGS Get contents and addresses of all save registers in stack
0358 730 :++
0358 731 : FUNCTIONAL DESCRIPTION:
0358 732 :
0358 733 : GET_REGS scans the stack and finds all registers saved
0358 734 : in call frames back to the signal facility. Thus it
0358 735 : makes an image of the registers at the time of the
0358 736 : exception or CALL LIB$SIGNAL/STOP. Because a double
0358 737 : operand may be saved in two different places, an image
0358 738 : array of addresses where the registers are saved is also created.
0358 739 : Note: GET_REGS assumes:
0358 740 : caller has saved R2:R11 in frame using its entry mask so all registers
0358 741 : are in memory somewhere. Stack scan is defensive against bad stacks.
0358 742 : Note: to reconstruct contents of SP at time of exception or call LIB$SIGNAL,
0358 743 : Use of the fact that the signal args list is pushed on stack first is made.
0358 744 : That is SP is = adr of last signal arg/ +4. Also depends on saved PC being
0358 745 : SYSSCALL_HANDL+4.
0358 746 :
0358 747 : CALLING SEQUENCE:
0358 748 :
0358 749 : JSB GET_REGS
0358 750 :
0358 751 : INPUT PARAMETERS:
0358 752 :
0358 753 : NONE
0358 754 :
0358 755 : IMPLICIT INPUTS:
0358 756 :
0358 757 : SIG_ARGS_ADR.(AP) : ADR. of array of signal args
0358 758 : MCH_ARGS_ADR.(AP) : ADR. of array of mechanism args
0358 759 :
0358 760 : OUTPUT PARAMETERS:
0358 761 :
0358 762 : NONE
0358 763 :
0358 764 : IMPLICIT OUTPUTS:
0358 765 :
0358 766 : REG_IMAGE(FP) : set reg image array R0:PC/PSL
0358 767 : ADR_IMAGE(FP) : Set adr where reg saved R0:PC/PSL
0358 768 : : except adr. where SP SAVED = 0, since not
0358 769 :
0358 770 : COMPLETION CODES:
0358 771 :
0358 772 : NONE JSB
0358 773 :
0358 774 : SIDE EFFECTS:
0358 775 :
0358 776 : If error, RET with error code
0358 777 : --
0358 778 :
0358 779 : +
0358 780 : Registers used:
0358 781 :
0358 782 : R0 = scratch
0358 783 : R1 = pointer to register image array (REG_IMAGE)
0358 784 : R2 = stack frame pointer
0358 785 : R3 = ADR. of register save area in frame
```



```
035B 786 : R4 = Loop count
035B 787 : R5 = pointer to address image array (ADR_IMAGE)
035B 788 : R6 = register save mask
035B 789 :-
035B 790
035B 791 GET_REGS: : get register image
035B 792
035B 793 :+
035B 794 : Setup loop to scan back through stack
035B 795 :-
035B 796
51 BC AD DE 035B 797 MOVAL REG_IMAGE(FP), R1 : R1 = Adr. reg image vector
52 5D D0 035F 798 MOVL FP, R2 : R2 = Adr. of current frame
0362 799 : where all callers register saved
54 01 10 78 0362 800 ASHL #16, #1, R4 : R4 = max loop count = 65K
55 FF78 CD DE 0366 801 MOVAL ADR_IMAGE(FP), R5 : R5 = adr. of array of address where
036B 802 : registers are saved.
036B 803 :+
036B 804 : Loop to scan call stack back to signal exception
036B 805 :-
036B 806
53 14 52 C1 036B 807 LOOP: ADDL3 R2, #SF$L_SAVE_REGS, - : stack frame adr + offset to first reg saved
036F 808 R3 : R3 = adr. of first saved reg.
036F 809 CLRL R0 : R0 = first possible register # saved
56 06 A2 0C 00 EF 0371 810 EXTZV #SF$V_SAVE_MASK, - : position of save mask
0377 811 #SF$S_SAVE_MASK, - : size of save mask
0377 812 SF$W_SAVE_MASK(R2), R6 : R6 = register save mask
0377 813
0377 814 :+
0377 815 : loop to copy saved registers R0:R11 from one call stack frame
0377 816 : to register image array also set address of register image array.
0377 817 :-
0377 818
56 0C 50 EA 0377 819 LOOP1: FFS R0, #12, - : find next register in saved bit mask
50 50 037B 820 R6, R0 : R0 = register number of next saved reg.
037C 821
037C 822 BEQL 10$, : branch if finished 12-bit reg mask
63 04 00 0D 037E 823 PROBEW #0, #4, (R3) : check if stack still writeable
24 13 0382 824 BEQL BAD_STACK1 : branch if stack bad
6540 53 D0 0384 825 MOVL R3, -(R5)[R0] : store address of where Rn saved
6140 83 D0 0388 826 MOVL (R3)+, (R1)[R0] : copy saved Rn to image + Rn
E7 56 50 E4 038C 827 BBSC R0, R6, LOOP1 : clear bit n for Rn, get next bit
0390 828
0390 829 :+
0390 830 : check if frame just saved is that of call to handler from signal or exception
0390 831 :-
0390 832
00000004'8F 10 A2 D1 0390 833 10$: CMPL SF$L_SAVE_PC(R2), - : saved PC the one from call to handler?
0398 834 #SYS$CALL_HANDL+4 : absolute system vector adr
16 13 0398 835 BEQL END_SCAN : branch if yes
039A 836
039A 837 :+
039A 838 : step (cautiously) to previous frame
039A 839 :-
039A 840
14 00 0D 039A 841 PROBEW #0, #SF$L_SAVE_REGS, - : check if fixed part of previous frame ok
OC B2 039D 842 #SF$L_SAVE_FP(R2) :
```

```
52 0C 07 13 039F 843 BEQL BAD_STACK1 ; branch if frame not writeable
    0C A2 D0 03A1 844 MOVL SF$C_SAVE_FP(R2), R2 ; R2 = adr. of previous frame
    C3 54 F5 03A5 845 SOBGTR R4, COOP ; go back if haven't scanned too many frames
    03A8 846
    03A8 847 ;+
    03A8 848 ; here if bad stack - return LIB$_BADSTA to caller of LIB$FLT_FIXUP
    03A8 849 ;+
    03A8 850 ;+
    03A8 851 BAD_STACK1:
    03A8 852 MOVL #LIB$_BADSTA, R0 ; R0 = BAD STACK completion code
    03AF 853 RET ; return to caller of LIB$FIXUP_FLT
    03B0 854 ; not JSB caller of GET_REGS
    03B0 855
    03B0 856 ;+
    03B0 857 ; Here when scanned all frames back to call to handler
    03B0 858 ; Copy R0:R1 from mechanism vector. Set AP,FP,SP,PC,PSL
    03B0 859 ; Also set address where each of these registers is saved
    03B0 860 ;+
    03B0 861
    03B0 862 END_SCAN:
50 08 AC D0 03B0 863 MOVL MCH_ARGS_ADR(AP), R0 ; R0 = adr. of signal mechanism arglist
    03B4 864
65 0C A0 DE 03B4 865 MOVAL CHF$MCH_SAVR0(R0), - ; adr. where R0 saved
    03B8 866 ; to vector of addresses
04 A5 10 A0 DE 03B8 867 MOVAL CHF$MCH_SAVR1(R0), - ; adr. where R1 saved
    03BD 868 ; to image address vector
61 0C A0 7D 03BD 869 MOVQ CHF$MCH_SAVR0(R0), - ; saved R0/R1
    03C1 870 ; to register image vector
    51 30 C0 03C1 871 ADDL #AP_OFF, R1 ; R1 = adr. in image vector of AP/FP
    55 30 C0 03C4 872 ADDL #AP_OFF, R5 ; R5 = adr. in image address vector of AP/FP
85 08 A2 DE 03C7 873 MOVAL SF$C_SAVE_AP(R2), - ; adr of saved AP
    03CB 874 ; to image address vector
85 0C A2 DE 03CB 875 MOVAL SF$C_SAVE_FP(R2), - ; adr of saved FP
    03CF 876 ; to image address vector
81 08 A2 7D 03CF 877 MOVQ SF$C_SAVE_AP(R2), - ; saved AP/FP
    03D3 878 ; to image register vector
50 04 BC 9A 03D3 879 MOVZBL @SIG_ARGS_ADR(AP), R0 ; R0 = # of signal args
50 04 BC 40 DE 03D7 880 MOVAL @SIG_ARGS_ADR(AP)(R0), -
    03DC 881 ; R0 = adr of last signal arg
    50 04 C0 03DC 882 ADDL #4, R0 ; R0 = SP at time of exception or call LIB$S
    03DF 883 ; NOTE: this a spec from LIB$SIGNAL and
    03DF 884 ; exception processing of operating system!!
    81 85 D4 03DF 885 CLRL (R5)+ ; SP not saved anywhere so set IMAGE_ADR TO
    81 50 D0 03E1 886 MOVL R0, (R1)+ ; set image SP
    81 70 7D 03E4 887 MOVQ -(R0), (R1)+ ; copy PC/PSL to image (always last
    03E7 888 ; 2 signal arguments)
85 80 DE 03E7 889 MOVAL (R0)+, (R5)+ ; set adr. where PC saved
85 80 DE 03EA 890 MOVAL (R0)+, (R5)+ ; set adr. where PSL saved
    03ED 891 ; return (to LIB$FIXUP_FLT)
    03EE 892
```

```
03EE 894 :++
03EE 895 : OUR_HANDLER - Local condition handler
03EE 896 :
03EE 897 : This condition handler is enabled by LIB$FIXUP_FLT. If the signal
03EE 898 : depth is zero and if the exception is SSS_ACCVIO or SSS_ROPRAND, then
03EE 899 : LIB$SIG_TO_RET is called to cause LIB$FIXUP_FLT to return the
03EE 900 : exception condition as a status. All other exceptions are resigalled.
03EE 901 :--
03EE 902 :
03EE 903 OUR_HANDLER:
03EE 904 .WORD 'M<>' ; Save nothing
50 0918 8F 0000 03F0 905 MOVZWL #SS$ RESIGNAL, R0 ; Resignal if nothing else
51 08 AC D0 03F5 906 MOVL CHF$C_MCHARGLIST(AP), R1 ; Get mechanism args list
08 A1 D5 03F9 907 TSTL CHF$L_MCH_DEPTH(R1) ; Is depth zero?
1D 12 03FC 908 BNEQ 90$ ; If not, resignal
51 04 AC D0 03FE 909 MOVL CHF$L_SIGARGLIST(AP), R1 ; Get signal args list
51 04 A1 D0 0402 910 MOVL CHF$L_SIG_NAME(R1), R1 ; Get signal name
OC 51 D1 0406 911 CMPL R1, #SS$_ACCVIO ; Access violation?
09 13 0409 912 BEQL 10$ ; If so, call LIB$SIG_TO_RET
00000454 8F 51 D1 040B 913 CMPL R1, #SS$_ROPRAND ; Reserved operand?
07 12 0412 914 BNEQ 90$ ; If not, resignal
00000000'GF 07 12 0412 914 BNEQ 90$ ; If not, resignal
04 0414 915 10$: CALLG (AP), G^LIB$SIG_TO_RET ; Unwind to LIB$FIXUP_FLT's caller
041B 916 90$: RET ; Return to CHF
041C 917
041C 918 .END ; end of LIB$FIXUP_FLT
```


LIB\$FIXUP FLT
Symbol table

- Fixup floating reserved operand M 3

16-SEP-1984 00:09:10 VAX/VMS Macro V04-00
6-SEP-1984 11:07:14 [LIBRTL.SRC]LIB\$FIXUPF.MAR;1Page 21
(7)

ADR_IMAGE	= FFFFFFFF78		
AP_OFF	= 00000030		
AUTO_DECR	000002B0	R	02
AUTO_INCR	000002B7	R	02
AUTO_INCR_DEF	000002C3	R	02
BAD_STACKT	000003A8	R	02
BPT	= 00000003		
BYTE_DISPL	000002D2	R	02
BYTE_DISPL_DEF	000002D6	R	02
CH\$C_MCHARGLST	= 00000008		
CH\$C_MCH_DEPTH	= 00000008		
CH\$C_MCH_SAVRO	= 0000000C		
CH\$C_MCH_SAVR1	= 00000010		
CH\$C_SIGARGLST	= 00000004		
CH\$C_SIG_NAME	= 00000004		
CHK_OP_RSB	00000306	R	02
DEST_D	00000227	R	02
DEST_F	00000222	R	02
DEST_G	00000232	R	02
DEST_H	0000023E	R	02
DISP	000002E4	R	02
DISPL_DEF	000002EE	R	02
DOUB_END	00000158	R	02
DOUB_TAB	000000A0	R	02
END_SCAN	000003B0	R	02
EXIT	0000024E	R	02
FIXUP	00000205	R	02
GET_REGS	0000035B	R	02
G_H	= 000000FD		
IMAGE_PC	= FFFFFFFF8		
INDEXED	00000289	R	02
K_SAV_IMAGE_SIZ	= 00000044		
LIB\$FIXUP_FCT	0000015F	RG	02
LIB\$GET_OPCODE	*****	X	00
LIB\$SIG_TO_RET	*****	X	00
LIB\$BADSTA	*****	X	00
LONG_DISPL	000002E2	R	02
LONG_DISPL_DEF	000002EC	R	02
LOOP	0000036B	R	02
LOOP1	00000377	R	02
LOOP_OP	0000025D	R	02
MCH_ARGS_ADR	= 00000008		
NEW_OPERAND	= 0000000C		
NEXT_BYTE	00000341	R	02
NEXT_LONG	00000352	R	02
NEXT_OPERAND	00000252	R	02
NEXT_OPERANDX	00000340	R	02
NEXT_WORD	00000349	R	02
NOT_RESRV	0000033E	R	02
OP_B	= 00000001		
OP_D	= 00000004		
OP_F	= 00000003		
OP_G	= 00000005		
OP_H	= 00000006		
OP_SIZES	00000158	R	02
OP_W	= 00000002		
OUR_HANDLER	000003EE	R	02

RO_OFF	= 00000000		
R1_OFF	= 00000004		
REG	00000291	R	02
REG_DEF	000002A9	R	02
REG_IMAGE	= FFFFFFFFBC		
RESIGNAL	000001EC	R	02
SCAN	000001F2	R	02
SEARCH	000001C2	R	02
SET_OA2	000002F7	R	02
SF\$C_SAVE_AP	= 00000008		
SF\$C_SAVE_FP	= 0000000C		
SF\$C_SAVE_PC	= 00000010		
SF\$C_SAVE_REGS	= 00000014		
SF\$C_SAVE_MASK	= 0000000C		
SF\$V_SAVE_MASK	= 00000000		
SF\$W_SAVE_MASK	= 00000006		
SIG_ARGS_ADR	= 00000004		
SING_END	000000A0	R	02
SING_TAB	00000000	R	02
SS\$ACCVIO	= 0000000C		
SS\$CONTINUE	= 00000001		
SS\$RESIGNAL	= 00000918		
SS\$ROPRAND	= 00000454		
ST\$S_COND_ID	= 00000019		
ST\$V_COND_ID	= 00000003		
SY\$CALL_HANDL	*****	X	00
S_FMZERO	= 00000009		
S_GMZERO	= 0000000C		
S_HMZERO	= 00000010		
V_FMZERO	= 00000007		
V_GMZERO	= 00000004		
V_HMZERO	= 00000000		
WORD_DISPL	000002DA	R	02
WORD_DISPL_DEF	000002DE	R	02

LIB
V03

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_LIB\$CODE	0000041C (1052.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.03	00:00:01.85
Command processing	103	00:00:00.31	00:00:02.19
Pass 1	233	00:00:04.26	00:00:18.30
Symbol table sort	0	00:00:00.53	00:00:01.85
Pass 2	169	00:00:01.40	00:00:06.12
Symbol table output	11	00:00:00.10	00:00:01.04
Psect synopsis output	3	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	550	00:00:06.65	00:00:31.37

The working set limit was 1500 pages.
37666 bytes (74 pages) of virtual memory were used to buffer the intermediate code.
There were 30 pages of symbol table space allocated to hold 559 non-local and 15 local symbols.
918 source lines were read in Pass 1, producing 15 object records in Pass 2.
12 pages of virtual memory were used to define 11 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	8

561 GETS were required to define 8 macros.
There were no errors, warnings or information messages.
MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:LIBFIXUPF/OBJ=OBJ\$:LIBFIXUPF MSRC\$:LIBFIXUPF/UPDATE=(ENHS\$:LIBFIXUPF)

0207 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

LIBFLTUND
LIS

LIBGETSYI
LIS

LIBICHAR
LIS

LIBINITIA
LIS

LIBEXUPF
LIS

LIBGETFOR
LIS

LIBGETINP
LIS

LIBINISHR
LIS

LIBGETDVI
LIS

LIBGETOPC
LIS

LIBFNDING
LIS

LIBGETMSG
LIS

LIBINDEX
LIS

LIBINSQHI
LIS

LIBGETUP1
LIS

LIBGETTAB
LIS